Projet L1, S2, 2015: Simulation de fourmis, Soutenance la semaine du 4 mai.

1 Introduction

On considère une grille de 20 lignes 20 colonnes. Une case de la grille peut être vide, ou contenir une et une seule des trois choses suivantes : une fourmi, du sucre ou un élément de nid. Une case ne peut pas contenir en même temps deux fourmis, ou bien du sucre et une fourmi, ou bien un élément de nid et une fourmi. Les fourmis doivent donc contourner le sucre et le nid, et se contourner entre elles. Une fourmi peut ramasser du sucre si elle se trouve à coté d'une case contenant du sucre. Une fourmi peut poser du sucre dans le nid si elle porte du sucre et se trouve à coté d'une case contenant un élément de nid.

Au début, les fourmis cherchent du sucre aléatoirement sur la grille. Une fois qu'une fourmi a trouvé du sucre, elle le ramène vers le nid en suivant des "phéromones de nid", qui indiquent le chemin de retour vers le nid. Pendant leur retour du tas de sucre vers le nid, les fourmis marquent leur chemin avec des "phéromones de sucre", qui vont permettre aux autres fourmis de trouver le sucre plus rapidement. Les phéromones sont des substances émises par la plupart des animaux et certains végétaux, et qui agissent comme des messagers sur des individus de la même espèce. Extrêmement actives, elles agissent en quantités infinitésimales, si bien qu'elles peuvent être détectés, ou même transportées, à plusieurs kilomètres.

Pour simplifier, nous supposerons que les "phéromones de nid" sont présentes en permanence et installent un gradient dirigé vers le nid. Cela signifie que l'intensité des phéromones augmente au fur et à mesure que l'on se rapproche du nid. Pour se rapprocher du nid, il suffit donc de se déplacer vers une case de plus forte intensité en phéromones. Les "phéromones de sucre", elles, s'évaporent. Ainsi, les fourmis ne s'obstineront pas à suivre un chemin de phéromones de sucre vers un tas de sucre qui a été complètement vidé.

1.1 Comportement des fourmis

Pour décrire le comportement des fourmis, nous utiliserons les prédicats (fonctions booléennes) listés ci-dessous.

- Prédicats sur une fourmi f :

`chercheSucre(`f`)`	f ne porte pas de sucre
`rentreNid(`f`)`	f porte du sucre

- Prédicats sur une place (une case) p :

`contientSucre(`p`)`	p contient du sucre
`contientNid(`p`)`	p contient un élément de nid
`contientFourmi(`p`)`	p contient une fourmi
`vide(`p`)`	p ne contient ni sucre, ni fourmi, ni élément de nid
`surUnePiste(`p`)`	l'intensité des phéromones de sucre sur p est non nulle

- Prédicats sur une place p_1 et une place voisine p_2 :

`plusProcheNid(`p_1`, `p_2`)`	l'intensité des phéromones de nid sur p_1 est plus importante que celle sur p_2
`plusLoinNid(`p_1`, `p_2`)`	l'intensité des phéromones de nid sur p_1 est moins importante que celle sur p_2

Pour une place donnée, plusieurs prédicats peuvent être vrais, par exemple: `surUnePiste(`p`)` et `vide(`p`)`.

Le principe général du comportement des fourmis est le suivant. Elles cherchent du sucre en bougeant aléatoirement. Quand elles trouvent de la phéromone de sucre, elles s'éloignent le

plus vite possible du nid, tout en restant sur la piste tracée par la phéromone de sucre. Quand elles sont à côté du sucre, elles chargent une partie du sucre, puis se rapprochent le plus vite possible du nid tout en posant des phéromones de sucre sur les cases empruntées.

Plus formellement, le comportement d'une fourmi est dicté par les règles énoncées ci-dessous par ordre de priorité décroissante.

Règles :

Soient f une fourmi, p_1 la case qu'elle occupe et p_2 une case adjacente à p_1.

1. Si `chercheSucre(`f`)` et `contientSucre(`p_2`)`,
   alors f charge du sucre et pose une phéromone de sucre sur p_1.

2. Si `rentreNid(`f`)` et `contientNid(`p_2`)`,
   alors f pose son sucre.

3. Si `rentreNid(`f`)` et `vide(`p_2`)` et `plusProcheNid(`p_2, p_1`)`,
   alors f se déplace sur p_2 et pose une phéromone de sucre sur p_2.

4. Si `chercheSucre(`f`)` et `surUnePiste(`p_1`)` et `vide(`p_2`)` et `plusLoinNid(`p_2, p_1`)` et `surUnePiste(`p_2`)`,
   alors f se déplace sur p_2.

5. Si `chercheSucre(`f`)` et `surUnePiste(`p_2`)` et `vide(`p_2`)`,
   alors f se déplace sur p_2.

6. Si `chercheSucre(`f`)` et `vide(`p_2`)`,
   alors f se déplace sur p_2.

1.2 Simulation

La simulation consiste essentiellement à initialiser la grille en plaçant les fourmis, le nid, le sucre et les phéromones de nid, puis à itérer un certain nombre de fois le déplacement des fourmis. Il faut aussi diminuer périodiquement l'intensité des phéromones de sucre pour tenir compte de leur évaporation.

Initialisation de la grille avec les phéromones de nid

Les phéromones de nid ont une intensité décrite par un réel compris entre 0 et 1. L'intensité vaut 1 sur le nid et diminue linéairement quand on s'éloigne du nid. Pour initialiser la grille avec les phéromones de nid, on commence par donner l'intensité 1 aux cases du nid et l'intensité 0 partout ailleurs. Ensuite, si la grille est constituée de t lignes et t colonnes, on itère t fois la règle suivante sur toutes les cases :

$$\text{Si } (p < 1) \text{ alors } p \leftarrow \max(m - \frac{1}{t}, 0)$$

avec p l'intensité des phéromones de nid sur la case considérée et m la valeur maximale des phéromones de nid sur les cases voisines.

Déplacement des fourmis

Pour chaque fourmi f sur une case p_1, on essaie d'appliquer les règles une par une, de la plus prioritaire à la moins prioritaire. Pour chaque règle, on cherche une case p_2 parmi les cases voisines de p_1 qui vérifient la condition d'application de règle. Si l'on trouve une telle case, on applique la règle. Sinon, on passe à la règle suivante. Si aucune règle ne s'applique, la fourmi

ne se déplace pas.

Remarque sur la règle 6 : si l'on parcourt le voisinage toujours dans le même sens pour trouver une case vide p_2 où déplacer la fourmi, on va beaucoup plus souvent selectionner p_2 parmi les cases observées en premier. La simulation ne paraitra alors pas très naturelle, les fourmis ne se déplaçant pas vraiment aléatoirement. Par exemple, si l'on cherche une case voisine vide en commençant par regarder la case au dessus, alors les fourmis se dirigeront très souvent vers le haut. Lors de l'application de la règle 6, vous devrez donc choisir la case p_2 aléatoirement parmi les cases voisines vides.

Phéromones de sucre

L'intensité des phéromones de sucre est un entier compris entre 0 et 255, qui diminue de 5 à chaque itération de la simulation. Lorsqu'une fourmi pose des phéromones de sucre, elle pose directement 255.

2 Affichage

On va générer une suite d'images au format "ppm" dont les noms seront de la forme f000, f001, f002 .... représentant chaque étape de la simulation. On va ensuite utiliser le logiciel *beneton movie* pour les rassembler en un fichier gif animé. Il devrait être installé au bâtiment 336 et est téléchargeable gratuitement à l'adresse `http://www.benetonsoftware.com/Beneton_Movie_GIF.php`.

Le format des fichiers ppm est donné en annexe, il contient essentiellement un tableau de couleurs des pixels. On représente chaque case de la grille par un pixel coloré. La couleur dépend du contenu de la case :

- les fourmis en noir
- le sucre en orange
- le nid en bleu
- la phéromone de sucre en rouge
- la phéromone de nid en vert

Si une case contient plusieurs éléments, on affiche en priorité l'élément qui est le plus haut dans la liste ci-dessus. L' amélioration suivante devra être apportée dans un deuxième temps: les phéromones seront affichées avec un dégradé permettant de rendre compte de leur intensité. Ainsi, les cases seront vert vif près du nid et de plus en plus sombre quand la phéromone de nid décroît. De même, les cases seront rouge vif quand une phéromone de sucre vient d'être déposée et de plus en plus sombre quand celle-ci s'évapore.

3 Implémentation de la simulation

Afin de vous aider à réaliser le projet, nous vous donnons une analyse de la suite d'actions à mettre en oeuvre pour implémenter la simulation.

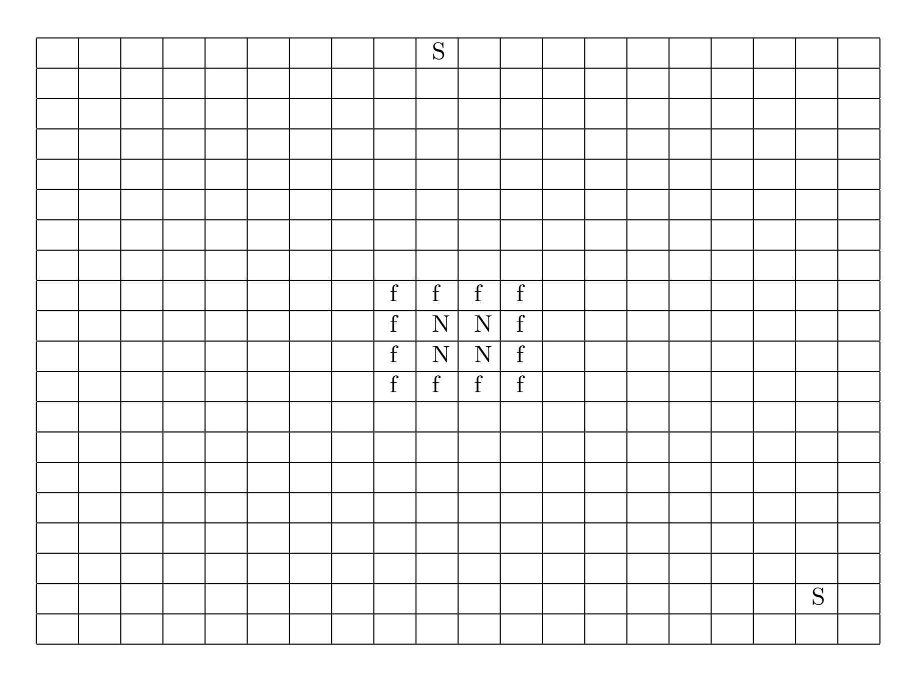
1. Spécifier **un type abstrait fourmi**, **un type abstrait place** (pour les cases) et **un type abstrait grille** contenant toutes les informations nécessaires pour la simulation.

   Aide : Le type abstrait ensemble de fourmis est utile car les traitements devront être faits sur chaque fourmi. Rassembler les fourmis dans un ensemble permet d'itérer sur les

fourmis en utilisant des boucles parcourant les éléments de l'ensemble. Néamnoins, il n'est pas nécessaire de programmer ce type abstrait, il suffit d'utiliser un vecteur de fourmis. Cette remarque est aussi valable pour manipuler des ensembles de coordonnées;

2. Décider des types concrets associés.
   Aide 1 : Une fourmi doit être associée à ses coordonnées dans la grille. Pour cela, utilisez le type concret `coord` défini dans le TP projet.
   Aide 2 : Comme une case peut contenir ou non une fourmi, il est recommendé d'associer à chaque place un booléen qui dit s'il y a une fourmi ou pas; Il n'est pas nécessaire de stocker un pointeur vers une fourmi dans une case occupée par une fourmi, car on n'a pas besoin de retrouver la fourmi depuis la case.
   Aide 3 : Le type concret de place peut stocker sa propre coordonnée, de sorte que lorqu'on déplace une fourmi, on a besoin seulement de passer deux places, et on peut lire la nouvelle coordonnée sur la case d'arrivée.

3. **Initialiser une grille** 20×20 avec au centre un nid de 2×2, 12 fourmis autour et deux tas de sucre loin du nid. L'intensité des phéromones de nid sera pour l'instant nulle partout. Le schéma suivant est un exemple de situation initiale :



   f = Fourmi, N= Nid, S= Sucre

4. Afficher le contenu de cette grille dans le terminal pour vérifier que l'initialisation a correctement été effectuée.

5. **Afficher cette grille** initiale dans un fichier ppm et le visualiser avec "benetton movie".

6. Écrire une procédure `deplacerFourmi` qui prend en paramètre une fourmi f, sa place p_1, une place p_2, et déplace f de p_1 vers p_2. Tester cette procédure en déplaçant une fourmi de la grille initiale.

7. Implementer le prédicat vide(p:place). À l'aide de la fonction `choisirCoordAleatoirement` du TP projet, déplacer chaque fourmi vers un de ses voisins vides et afficher la grille à nouveau.

8. Itérer 30 fois ce comportement et **générer un premier film** rassemblant les 30 images produites.

9. Initialiser la grille avec **l'intensité des phéromone de nid** associée à chaque case. Afficher l'intensité en phéromone de nid des cases de la grille dans le terminal pour vérifier que l'initialisation a correctement été effectuée.

10. Faire en sorte que dans les images générées, les cases vides soient affichées en dégradé de vert représentant l'intensité des phéromones de nid.

11. **Implémenter les prédicats** définis dans la section 1.1 comme des fonctions `cond`*i* retournant vrai si la condition de la règle i est vérifiée.

12. **Implémenter les règles 1, 2, 3 et 6** et déplacer les fourmis en suivant ces règles, sur 30 itérations. On doit observer épisodiquement quelques fourmis qui ramènent du sucre au nid.

13. **Rajouter les règles 4 et 5** permettant d'accélérer "exponentiellement" le rapatriement du sucre. Construire un film d'environ 300 itérations.

14. (optionnelle) A présent, lorsque les fourmis ramènent du sucre, le tas de sucre doit diminuer. Les fourmis peuvent vider successivement plusieurs tas de sucre, sans modifier les règles, du fait que le tas de sucre diminue lorsque les fourmis en chargent.

4 Organisation et évaluation du projet

Deux séances de TD et deux séances de TP porteront sur le projet. N'hésitez pas à poser des questions à vos encadrants de TP pour régler les problèmes que vous ne parvenez pas à surmonter. Il est indispensable de commencer à travailler sur le projet dès maintenant.

L'évaluation du projet se fera au travers d'une soutenance orale prévue la semaine du 4 mai. L'organisation de la soutenance est la suivante:

- vous vous installez sur une machine, chargez votre programme et vérifiez rapidement que tout fonctionne comme la dernière fois que vous l'avez testé;

- les deux membres du binôme ont 10 mn pour présenter ensemble le résultat de leur travail (organisation du code, visualisation des résultats, discussion sur les problèmes rencontrés...);

- ensuite l' examinateur a 5 mn pour juger de votre maîtrise de la programmation. À partir de ce moment, vous serez considérés individuellement, et un effort important sera fait pour mettre en avant clairement votre implication dans le projet. Un étudiant ne peut prétendre être noté sur un code qu'il ne réussit pas à expliquer. Un étudiant absent à la soutenance aura zéro sur vingt.

Le principal critère d'évaluation est le nombre d'actions fonctionnant correctement. Nous prendrons également en compte :

- Qualité de présentation...

- Types abstraits : pertinence, choix alternatifs, utilisation dans le code...

- Qualité du code : modularité, lisibilité...

- Maîtrise du code;

- Originalité : gestion de problèmes rencontrés, affichage d'informations intermédiaires...

5 TD Types abstraits du Projet

Nous allons étudier les types abstraits découlant d'une analyse du projet de simulation de fourmis. Les questions qui suivent ont pour but de vous aider dans la réalisation du projet. Nous allons considérer quatre types abstraits : un type représentant une fourmi, un type représentant un ensemble de fourmis, un type représentant une place (c'est à dire une case) et un type représentant la grille.

5.1 Les coordonnées

On suppose que l'on dispose des types abstraits **coord** et **ensCoord** que vous avez implémentés dans le TP projet. Les fonctions utiles au présent TD seront :

- fonction **creerCoord** (Donnée x : entier, Donnée y : entier) → **coord**
  { crée la coordonnée (x,y) }
- fonction **trouverVoisins**(Donnée c : **coord**) → **ensCoord**
  { retourne l'ensemble des coordonnées voisines à la coordonnée c }

5.2 Les fourmis

Le type abstrait **fourmi** représente une fourmi. Il est spécifié de la manière suivante :

- Constructeur :

  fonction **creerFourmi** (________________) → _______
  { crée une fourmi ne portant pas de sucre, à partir de ses coordonnées sur la grille }

- Accès :

  fonction **lireCoord** (________________) → ______
  { retourne les coordonnées de la fourmi }

- Prédicats :

  fonction **chercheSucre** (________________) → _______
  { retourne vrai si la fourmi cherche du sucre }

  fonction **rentreNid** (________________) → _______
  { retourne vrai si la fourmi rentre au nid }

- Modifications :

  procédure **dechargerSucre** (_________________________)
  { supprime la charge de sucre portée par la fourmi }

  procédure **chargerSucre** (_________________________)
  { ajoute une charge de sucre sur la fourmi }

1. Complétez les spécifications ci-dessus en remplissant les trous dans les entêtes.

Le type abstrait **ensFourmis** représente un ensemble de fourmis. Voici un extrait de ses spécifications :

- Constructeur :

  procédure **chargerEnsFourmis** (________________________ , ____________________)
  { crée un ensemble de fourmis positionnées aux coordonnées
  spécifiées par l'ensemble de coordonnées fourni en 2ème paramètre }

2. Complétez les spécifications ci-dessus en donnant les entêtes des procédures et fonctions.

5.3 La grille

Le type abstrait **place** représente une case de la grille. Il est spécifié de la manière suivante :

- Constructeur :

  procédure **creerPlaceVide** (______________________)
  { crée une place dont les coordonnées sont fournies en paramètre,
  ne contenant pas de fourmi, ni de sucre, ni d'élément de nid }

- Accès :

  fonction **lirePheroNid** (_______________) → ___
  { retourne l'intensité en phéromone de nid sur la place }

  fonction **lirePheroSucre** (_______________) → _____
  { retourne l'intensité en phéromone de sucre sur la place }

- Prédicats sur une place :

  fonction **contientFourmi** (_______________) → ______
  { retourne vrai si la place contient une fourmi }

  fonction **contientSucre** (_______________) → ______
  { retourne vrai si la place contient du sucre }

  fonction **contientNid** (_______________) → ______
  { retourne vrai si la place contient un élément de nid }

  fonction **surUnePiste** (_______________) → ______
  { retourne vrai si la place se trouve sur une piste vers du sucre
  (tracée par la présence de phéromones de sucre) }

- Modifications sur une place :

  procédure **poserSucre** (______________________)
  { pose du sucre sur la place }

  procédure **poserNid** (______________________)
  { pose un élément de nid sur la place }

  procédure **poserFourmi** (______________________)
  { pose une fourmi sur la place }

  procédure **poserPheroNid** (______________________ , _____________)
  { pose sur la place une phéromone de nid d'intensité fournie en 2ème paramètre }

  procédure **poserPheroSucre** (______________________)
  { pose une phéromone de sucre d'intensité maximale sur la place }

  procédure **diminuerPheroSucre** (______________________)
  { diminue l'intensité en phéromone de sucre sur la place }

- Modification sur deux places :

  procédure **deplacerFourmi** (______________________ ,
  ______________________ ,
  ______________________)
  { déplace la fourmi fournie en 1er paramètre qui est sur la place fournie en 2ème paramètre
  vers la place fournie en 3ème paramètre. (Il faut mettre à jour les coordonnées de la fourmi.)
  }

3. Complétez les spécifications ci-dessus en donnant les entêtes des procédures et fonctions.

4. Réalisez un prédicat **vide** qui prend une place en paramètre et renvoie vrai si la place ne contient ni du sucre, ni un élément de nid, ni une fourmi.

5. Réalisez un prédicat **plusProcheNid** qui prend renvoie vrai si la place fournie en 1er paramètre est plus proche du nid que la place fournie en 2ème paramètre.

Remarque : un prédicat **plusLoinNid** peut être réalisé suivant le même principe.

Le type abstrait **grille** représente la grille. Voici un extrait de ses spécifications :

- Constructeur :

  procédure **chargerGrilleVide** (Résultat g : **grille**)
  { initialise la grille avec une grille vide, peut se faire en itérant creerPlaceVide sur toutes les places de la grille }

- Accès et modifications :

  procédure **chargerPlace** (Donnée g : **grille**, Donnée c : **coord**, Résultat p : **place**)
  { donne en résultat la place de la grille située aux coordonnées indiquées }

  procédure **rangerPlace** (Donnée-Résultat g : **grille**, Donnée p : **place**)
  { met la place dans la grille. (Si on a stocké la coordonnée de la place, dans la place, ici, il n'y a pas besoin de passer de paramètre coordonnée, contrairement à chargerPlace). }

  Remarque : Lorqu'on charge une place, et qu'on la modifie, il ne faut pas oublier de la ranger dans la grille, sinon la modification ne sera pas enregistrée sur le terrain.

6. Réalisez une procédure **placerNid** qui place des éléments de nid sur la grille passée en premier paramètre, à l'ensemble des coordonnées fournies en 2ème paramètre.
   NB : On dispose pour cela de la structure de contrôle suivante pour itérer sur toutes les coordonnées `c` d'un ensemble de coordonnées `ec` :

```
pour chaque c dans ec faire
  ...
finpour
```

   Remarque : une procédure **placerSucre** peut-être réalisée suivant le même principe

7. Réalisez une procédure **placerFourmis** qui prend une grille et un ensemble de fourmis en paramètres, et pose l'ensemble de fourmis sur la grille.
   NB : On dispose pour cela de la structure de contrôle suivante pour itérer sur toutes les fourmis `f` d'un ensemble de fourmis `ef` :

```
pour chaque f dans ef faire
  ...
finpour
```

8. Réalisez une procédure **lineariserPheroNid** qui prend une grille en paramètre et linéarise l'intensité en phéromones de nid des cases de la grille (voir la formule dans l'énoncé du projet). On supposera que dans la grille passée en paramètre, l'intensité en phéromones de nid est maximale sur les places contenant un élément du nid et nulle partout ailleurs.

9. Réalisez une procédure **initialiserGrille** qui prend en paramètre une grille, un ensemble de fourmis, l'ensemble des coordonnées des éléments de sucre et l'ensemble des coordonnées des éléments de nid, et initialise la grille en plaçant les fourmis, le sucre et le nid et en linéarisant les phéromones de nid

10. Réalisez une procédure **diminuerPheroSucreGrille** qui diminue l'intensité des phéromones de sucre sur la grille pour modéliser leur évaporation.

6 TP projet: Types coordonnées et ensemble de coordonnées

Le code écrit dans ce TP sera utilisé dans le projet. Il est donc impératif de le tester de manière très poussée pour ne pas introduire de bug dans le projet.

On cherche à manipuler des coordonnées dans une grille, c'est à dire la paire constituée d'un numéro de ligne et d'un numéro de colonne.

1. Coder le type produit `coord`.

2. Coder la fonction `creerCoord` qui prend en paramètre un numéro de ligne *lig* et un numéro de colonne *col* et retourne une variable de type `coord` initialisée à la coordonnée (lig, col).

3. Coder la procédure `afficherCoord` qui prend en paramètre une valeur de type `coord` et affiche cette valeur sous la forme : (lig, col).

4. Tester ces fonctions et procédures en utilisant le programme principal suivant :

```
int main(){
  coord c1 = creerCoord(2,1);
  afficherCoord(c1);
  cout << endl;
  return 0;
}
```

5. Coder la fonction `coordonneesEgales` retournant vrai si deux coordonnées sont égales. Tester cette fonction avec le programme principal suivant :

```
int main(){
  coord c1 = creerCoord(2,1);
  coord c1_bis = creerCoord(2,1);
  coord c2 = creerCoord(3,4);

  // premier test
  afficherCoord(c1);
  if (coordonneesEgales(c1,c1_bis)) cout << "= "; else count << "<> ";
  afficherCoord(c1_bis);
  cout << endl;
  // second test
  afficherCoord(c1);
  if (coordonneesEgales(c1,c1_bis)) cout << "= "; else count << "<> ";
  afficherCoord(c2);
  cout << endl;

  return 0;
}
```

6. On souhaite maintenant représenter un ensemble d'au maximum `NB_COORD` coordonnées. Coder le type `ensCoord` avec un type produit.

7. Coder la procédure `afficheEnsCoord` qui affiche un ensemble de coordonnées.

8. Coder la fonction `creerEnsCoordVide` qui renvoie un ensemble de coordonnées ne contenant aucun élément.

9. Coder la procédure `ajouterCoord` qui prend en paramètre un ensemble de coordonnées *ec*, une coordonnée *c* et ajoute *c* à l'ensemble *ec*.

10. Tester ces procédures et fonctions avec le programme principal suivant :

```
int main(){
  coord c1 = creerCoord(2,1);
  coord c2 = creerCoord(3,4);
  coord c3 = creerCoord(0,0);

  // on construit un exemple d'ensemble
  ensCoord exemple = creerEnsCoordVide();
  ajouterCoord(exemple, c1);
  ajouterCoord(exemple, c2);
  ajouterCoord(exemple, c3);
  afficherEnsCoord(exemple);

  // on ajoute encore un element
  cout << "ajout d'un element :" << endl;
  ajouterCoord (exemple, creerCoord(4,0));
  afficherEnsCoord(exemple);

  return 0;
}
```

11. On s'intéresse maintenant à trouver les coordonnées des voisins d'une case dans une grille de `TAILLE` lignes et `TAILLE` colonnes. Par exemple, considérons la grille ci-dessous (`TAILLE` $= 5$) :

$(0,0)$	$(0,1)$	$(0,2)$	$(0,3)$	$(0,4)$
$(1,0)$	$(1,1)$	$(1,2)$	$(1,3)$	$(1,4)$
$(2,0)$	$(2,1)$	$(2,2)$	$(2,3)$	$(2,4)$
$(3,0)$	$(3,1)$	$(3,2)$	$(3,3)$	$(3,4)$
$(4,0)$	$(4,1)$	$(4,2)$	$(4,3)$	$(4,4)$

- Les voisins de $(2,1)$ sont $(1,0)$, $(1,1)$, $(1,2)$, $(2,2)$, $(3,2)$, $(3,1)$, $(3,0)$, $(2,0)$.
- Les voisins de $(3,4)$ sont $(2,3)$, $(2,4)$, $(3,3)$, $(4,3)$, $(4,4)$.
- Les voisins de $(0,0)$ sont $(0,1)$, $(1,0)$, $(1,1)$.

Coder la fonction `trouverVoisins` qui prend en paramètre une coordonnée c et retourne l'ensemble des coordonnées des voisins de c.

Si lig est la ligne de c et col est la colonne de c, on peut collecter les coordonnées des voisins de c dans un ensemble ev de la manière suivante :

pour i allant de i_{min} à i_{max} faire
    pour j allant de j_{min} à j_{max} faire
        si $(i,j) \neq (lig, col)$ alors
            ajouterCoordonnees $((i,j),\ ev)$
        finsi
    finpour
finpour

avec :
$$\begin{aligned} i_{min} &= max(lig-1,\ 0) \\ i_{max} &= min(lig+1,\ TAILLE) \\ j_{min} &= max(col-1,\ 0) \\ i_{max} &= min(col+1,\ TAILLE) \end{aligned}$$

Ajouter les instructions suivantes à la fin du programme principal précédent pour tester la fonction (en prenant soin d'adapter les noms des champs du type `ensCoord` aux noms précédement choisis).

```
  // on teste la fonction trouverVoisins
  ensCoord voisins;
  int i;
```

```
for (i = 0; i < exemple.nbElts; i++){
  voisins = trouverVoisins(exemple.tab[i]);
  cout << "les voisins de ";
  afficherCoord(exemple.tab[i]);
  cout << "sont ";
  afficherEnsCoord(voisins);
}
```

12. Coder une fonction `choisirCoordAleatoirement` qui prend un ensemble de coordonnées en paramètre et retourne une coordonnée au hasard parmi cet ensemble.
    Aide : la fonction `rand()` retourne un nombre aléatoire. Pour obtenir un nombre aléatoire entre 0 et n (compris), il faut donc appliquer un modulo $(n + 1)$ au résultat de l'appel à `rand()`.

    Ajouter un appel à cette fonction à la fin du programme principal pour la tester.

7 TD analyse descendante du Projet

Dans le TD précédent, nous avons effectué une analyse ascendante du projet. En effet, nous avons listé les opérations élémentaires nécessaires à la simulation des fourmis, et nous avons utilisé ces opérations basiques pour construire des opérations plus compliquées, elles-mêmes pouvant êtres combinées à nouveau.

Dans ce TD, nous allons adopter l'approche inverse, c'est à dire effectuer une analyse descendante du problème. Cela consiste à considérer le problème dans son ensemble, et le diviser en plusieurs sous-problèmes plus simples. Ces derniers peuvent être résolus indépendamment les uns des autres (éventuellement par des programmeurs différents) et peuvent être eux-mêmes divisés en sous-problèmes. Cette analyse descendante peut-être menée jusqu'à l'atteinte des sous-problèmes déjà traités dans le TP projet et le TD précédent.

Nous pourrons utiliser toutes les procédures et fonctions définies dans le TP projet et le TD précédent. Il est donc nécessaire de les avoir sous les yeux, ainsi que l'énoncé du projet.

7.1 Algorithme principal de la simulation

1. Réalisez l'algorithme principal de la simulation en supposant disponibles les procédures spécifiées ci-dessous.

- procédure **initialiserEmplacements** (Résultat lesFourmis : **ensFourmis**,
Résultat leSucre : **ensCoord**,
Résultat leNid : **ensCoord**)
{ Initialise les emplacements des fourmis, du sucre et du nid }
- procédure **dessinerGrille** (Donnée laGrille : **grille**)
{ Dessine une image de la grille dans un fichier }
- procédure **mettreAJourEnsFourmis** (Donnée-Résultat laGrille : **grille**,
Donnée-Résultat lesFourmis : **ensFourmis**)
{ déplace toutes les fourmis en appliquant les règles de leur comportement }

Le problème de la simulation a maintenant été divisé en sous-problèmes dont certains ont été traités dans le TD type abstrait, et d'autres sont traités par les trois procédures ci-dessus. Dans la suite du TD, nous traitons le sous-problème le plus difficile, à savoir la mise à jour des positions de l'ensemble des fourmis.

7.2 Déplacement des fourmis

2. Réalisez la procédure **mettreAJourEnsFourmis** utilisée dans l'algorithme principal, en supposant disponible la procédure spécifiée ci-dessous.

- procédure **mettreAJourUneFourmi** (Donnée-Resultat laGrille : **grille**, Donnée-Résultat uneFourmi : **fourmi**) { déplace une fourmi en appliquant les règles de comportement des fourmis }

NB : On dispose pour cela de la structure de contrôle suivante pour itérer sur toutes les fourmis `f` d'un ensemble de fourmis `ef` :

```
pour chaque f dans ef faire
  ...
finpour
```

3. Réalisez la procédure **mettreAJourUneFourmi**, en supposant disponibles la fonction et la procédure spécifiées ci-dessous.

- fonction **condition_n** (Donnée x : entier, Donnée f : **fourmi**, Donnée p1 : **place**, Donnée p2 : **place**) → booléen { retourne Vrai si la condition de déplacement de la règle numéro x s'applique sur la fourmi f, située sur la place p1, pour un déplacement vers la place p2 } - procédure **action_n** (Donnée x : entier, Donnée-Résultat f : **fourmi**, Donnée-Résultat p1 : **place**, Donnée-Résultat p2 : **place**) { applique la procédure de déplacement de la règle numéro x sur la fourmi f, située sur la place p1, pour un déplacement vers la place p2 }

NB: On suposera aussi disponibles : la fonction **cardinal** qui retourne le nombre d'éléments d'un ensemble de coordonnées `ec` passé en paramètre, ainsi que la fonction **lireElement** qui prend en paramètres un ensemble de coordonnées `ec` et un entier `i` et retourne le `i`-ème élément de `ec`.

4. Adaptez votre algorithme en prenant en compte la remarque sur la règle 6 dans l'énoncé du projet. On supposera disponible la fonction spécifiée ci-dessous.

fonction **voisinVideAleatoire** (Donnée g : **grille**, Donnée c : **coord**) → **coord** { retourne les coordonnées dans la grille g d'un voisin vide de la case de coordonnées c, choisi aléatoirement }

7.3 Règles de déplacement

5. Réalisez la fonction **condition_n** et la procédure **action_n** en supposant disponibles les fonctions **condition1**, ... , **condition6** qui correspondent aux conditions des règles 1 à 6 et les procédures **action1**, ... , **action6** qui correspondent aux actions des règles 1 à 6.

fonction **condition1** (Donnée f : **fourmi**, Donnée p1 : **place**, Donnée p2 : **place**) → booléen { retourne Vrai si la condition d'application de la règle de déplacement 1 est vérifiée } ... procédure **action1** (Donnée-Résultat f : **fourmi**, Donnée p1-Résultat : **place**, Donnée-Résultat p2 : **place**) { applique la règle de déplacement 1 } ...

8 Annexe

Un fichier contient une image décrite par la suite de caractères suivante : 1'- Un "magic number" identifiant le type. Pour un fichier ppm il s'agit des deux caractères "P3". 2- Des espaces (blancs, TABs, CRs, LFs). 3- Une largeur L, formatée avec des caractères ASCII en décimal. 4- Des espaces. 5- Une hauteur H, également en ASCII en décimal. 6- Des espaces. 7- La valeur de couleur maximale, également en ASCII en décimal. Comprise entre 1 et 65536. 8- Un retour à la ligne ou d'autres caractères espace. 9- Un tableau de H rangées, allant de bas en haut, chaque rangée contient L pixels de gauche à droite. Chaque pixel est un triplet des composantes RGB (c'est à dire rouge, vert, bleu) de la couleur, dans cet ordre. Tous les nombres doivent être précédés et suivis d'un espace. Les lignes ne doivent pas dépasser 70 caractères.

Voici l'exemple d'un fichier img000.ppm contenant une image de 16 pixels (4 par 4), représentant une diagonale rouge sur fond vert (les composantes RGB de la couleur rouge sont (255,0,0) et celles de la couleur verte sont (0,255,0)).

```
P3
4 4
255
255 0 0    0 255 0    0 255 0    0 255 0
0 255 0    255 0 0    0 255 0    0 255 0
0 255 0    0 255 0    255 0 0    0 255 0
0 255 0    0 255 0    0 255 0    255 0 0
```

Ce fichier ppm a été généré par le programme suivant :

```
#include <iostream>     // pour cout
#include <iomanip>      // pour setfill, setw
#include <sstream>      // pour ostringstream
#include <fstream>      // pour ofstream
#include <string>

using namespace std;
// variable globale permettant de creer des noms de fichiers differents
int compteurFichier = 0;
// action dessinant un damier
void dessinerDamier(){
  int i,j;
  int r,g,b;
  ostringstream filename;
  // creation d'un nouveau nom de fichier de la forme img347.ppm
  filename << "img" << setfill('0') << setw(3) << compteurFichier << ".ppm";
  compteurFichier++;
  cout << "Ecriture dans le fichier : " << filename.str() << endl;
  // ouverture du fichier
  ofstream fic(filename.str(), ios::out | ios::trunc);
  // ecriture de l'entete
  fic << "P3" << endl
      << 4 << " " << 4 << " " << endl
      << 255 << " " << endl;
  // ecriture des pixels
  for (i = 0; i < 4; i++){
      for (j = 0; j < 4; j++){
        // calcul de la couleur
        if (i == j) { r = 255; g = 0; b = 0; }
        else { r = 0; g = 255; b = 0; }
        // ecriture de la couleur dans le fichier
        fic << r << " " << g << " " << b << "    ";
      }
```

```
    // fin de ligne dans l'image
    fic << endl;
  }
  // fermeture du fichier
  fic.close();
}
int main (){
  dessinerDamier();
  return 0;
}
```

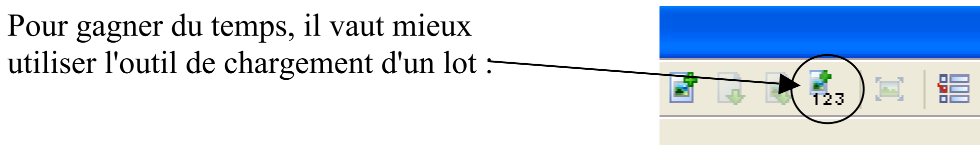
Mode d'emploi de Beneton Movie GIF

Le raccourci est dans le dossier Outils sur le Bureau

Au lancement un message apparaît ; c'est normal : les sons ne sont pas pris en charge !

Chargement d'un lot d'images

Pour gagner du temps, il vaut mieux utiliser l'outil de chargement d'un lot :



On doit alors remplir les différentes zones de saisie :

What is the filename prefix of your files? The number of the frame must follow that string. Put nothing here if your frames are just named by a number.

le radical des fichiers — Par exemple fic si les fichiers commencent tous par fic

Put the extension of your files. Don't forget the leading dot!

.ppm

Enter the minimum number of digits after the filename prefix.

3 — le nombre de chiffres qui suivent le radical (3 si les fichiers sont du genre fic000.ppm fic001.ppm etc.)

Load frames 0 to 100

Ex.: le radical des fichiers000.ppm, ... le radica...

Where are your files located?

Browse for folder

Z:\ — Le dossier où se trouvent les fichiers ppm

OK Cancel

Choix de la vitesse d'animation

Sélectionner toutes les images : un clic sur une des images puis Ctrl A
Préciser un délai (zone Delay), en centièmes de secondes
Créer le fichier de synthèse : bouton sauvegarde ou Ctrl S (on peut cocher Loop ou non)

Visualisation de l'animation

En restant dans Beneton Movie GIF : bouton Play . Utiliser si nécessaire les boutons de zoom.
On peut aussi faire glisser le fichier gif obtenu dans le programme GifZoom qui se trouve lui aussi dans le dossier Outils, ou lancer GifZoom puis ouvrir le fichier gif.
L'avantage de cette deuxième méthode est que le facteur de zoom est conservé pour la prochaine visualisation.